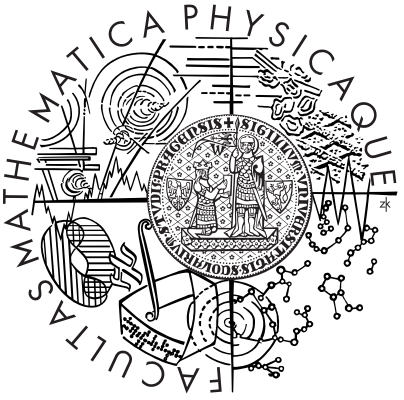
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

# BAKALÁRSKA PRÁCA




Lukáš Zemčák

## ACT-R v Pogamute

Katedra Software a Výuky Informatiky

Vedúci bakalárskej práce: Mgr. Cyril Brom, Ph.D.
Študijný program: Obecná informatika

2009

Ďakujem svojmu vedúcemu Mgr. Cyril Bromovi, Ph.D. za ponúknutú tému a množstvo zaujímavého materiálu. Mojim rodičom za trpezlivosť a podporu pri štúdiu. Manželke Lenke za podporu a syntaktické korektúry. Týmu Pogamutu za skvelú prácu a rýchle riešenie problémov. Petrovi Tóthovi za pripomienky a opravy všetkého druhu a Matejovi Vitáskovi za pomoc s úskaliami TEXu a anglickým abstraktom.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičaním práce a jej zverejňovaním.

V Prahe dňa 6.8.2009 Lukáš Zemčák

# Obsah

Názov práce: ACT-R v Pogamute
Autor: Lukáš Zemčák
Katedra: Katedra Software a Výuky Informatiky
Vedúci bakalárskej práce: Mgr. Cyril Brom, Ph.D.
E-mail vedúceho: brom@ksvi.mff.cuni.cz

Abstrakt: Akým spôsobom je mozog organizovaný tak, že dovoľuje vedomie? Adaptive Control of Thought-Rational (ACT-R) je kognitívna architektúra snažiaca sa odpovedať na túto otázku. Je inšpirovaná Fodorovou modulárnou architektúrou a súčasným pokrokom na poli kognitívnej neurovedy. Často je výhodné skúmať správanie kognitívnych architektúr na virtuálnych bytostiach. Pogamut je platforma, ktorá je navrhnutá na zjednodušenie vytvárania a ladenia umelých bytostí. Pogamut používa hru Unreal Tournament 2004 ako virtuálny svet. Prvá časť práce sa zaoberá teóriou ACT-R a jej vysvetlením ľudskej mysle, druhá predstavuje Pogamut a tretia hovorí o implementácii ACT-R rozšírenia do Pogamutu.

Kľúčové slová: kognitívna architektúra, ACT-R, Pogamut, UT2004

Title: ACT-R in Pogamut
Author: Lukáš Zemčák
Department: Department of Software and Computer Science Education
Supervisor: Mgr. Cyril Brom, Ph.D.
Supervisor's e-mail address: brom@ksvi.mff.cuni.cz

Abstract: How can be brain organized to enable cognition? Adaptive Control of Thought–Rational (ACT-R) is a cognitive architecture, trying to answer that question. It is inspired by Fodor modular architecture of mind and current advancements in field of cognitive neuroscience. It is often useful to investigate behavior of cognitive architectures on virtual beings. Pogamut is a platform project designed to simplify creation and tuning of artificial beings. Pogamut is using game Unreal Tournament 2004 as its virtual world. First part of work deals with ACT-R theory and its explanation of human mind, second part introduces Pogamut and third describes implementation of ACT-R plugin into Pogamut.

Keywords: Cognitive architecture, ACT-R, Pogamut, UT2004

# Kapitola 1

# Úvod

Vývoj riadenia virtuálnych agentov rýchlo napreduje. Tieto riadiace systémy sú písané pre rôzne platformy. Často je to platforma, ktorú preferuje ich autor. Pre experimentovanie s takto riadeným systémom je potrebný simulátor virtuálneho sveta napojiteľný na danú platformu. Nedostatok vhodných simulátorov vedie k zbytočnému vytváraniu "jednorázových" jednoduchých simulátorov špecifických pre daný experiment. Toto riešenie je postačujúce, ak je experiment jednoduchý. Problém nastáva ak experimentátor potrebuje komplexný simulátor virtuálneho sveta. Vtety musí siahnuť po externom viruálnom svete a naimplementovať spojenie s jeho riadiacim systémom.

Pre spojenie s virtuálnym svetom je možné použiť Pogamut. Je to projekt vytvorený s cieľom ponúknuť implementátorom agentov platformu, ktorá im zjednoduší spojenie so simulátorom a odtieni ich od technických detailov tohoto spojenia. V čase písania tejto práce Pogamut ponúkal pripojenie k virtuálnemu svetu hry Unreal Tournament 2004 (UT2004). Pogamut je navrhnutý tak, aby sa dal jednoducho pripojiť na iný simulátor a to bez zmeny agentovho rozhrania.

Riadenie virtuálnych agentov sa často opiera o poznatky z kognitívnej psychológie. Je to oblasť psychológie, ktorá sa zaoberá poznávacími procesmi ľudskej mysle. Ako sú: vnímanie, predstavivosť, myslenie, reč, pamäť a učenie. O týchto procesoch existujú rôzne teórie spolu s ich výpočtovými modelmi. Správanie niektorých modelov je výhodné skúmať vo virtuálnych prostrediach, či už kvôli interakcii s prostredím, alebo pre porovnanie so správaním reálnych ľudí.

Jednou z takýchto komplexných teórii je Adaptive Control of Thought-Rational (ACT-R), ktorá hovorí o štruktúre a kognitívnych procesoch v mozgu. Súčasťou ACT-R je aj výpočtový model a jednoduchý simulátor, ktorý však neobsahuje virtuálny svet. Ak by existoval virtuálny svet, v ktorom by sa dalo pracovať s ACT-R modelmi, odpadla by kognitívnym výskumníkom časť ich práce a ušetrili by zbytočný čas venovaný implementácii.

Tento problém sa pokúša vyriešiť PoJACT-R[1], ktorý rozširuje Pogamut o jACT-R, čo je Java implementácia simulátora pre ACT-R modely. PoJACT-R cez jACT-R a Pogamut vytvára prepojenie medzi ACT-R modelom a UT2004.

Táto bakalárska práca sa zaoberá práve realizáciou tohto prepojenia a snaží sa riešiť otázky s tým spojené:

1. Ako zachovať plausibilitu ACT-R
2. Ako zjednodušiť použitie pri zachovaní komplexnosti
3. Ako čo najviac využiť typovú bezpečnosť jazyka Java
4. Ako oddeliť jadro prepojenia a časťi závislé na UT2004

Podobnou prácou je napríklad prepojenie hry Quake a kognitívnej architektúry SOAR od Duchi a Laird [1]. Vo svojej práci tvrdia, že je dobré používať virtuálnu realitu pre testovanie výpočtových modelov agentov. ACT-R na riadenie agentov sa používa napríklad v systéme MOUT [2]. Práve MOUT je veľkou inšpirácou PoJACT-R, pretože rieši model komplexného virtuálneho agenta.

Táto práca je rozdelená na 6 kapitol. Po úvode nasleduje kapitola venovaná teórii ACT-R a jej prerekvizitám, ktoré sú potrebné pre pochopenie PoJACT-R a mala by slúžiť aj ako odrazový mostík ku komplexnejším prácam o kognícii a ACT-R [4], [5], [6], [7]. Tretia kapitola predstavuje Pogamut a jeho architektúru. Štvrtá kapitola sa zaoberá riešením problémov spojených s implementáciou. Piata kapitola hovorí o výkone a možnostiach

[1] Zloženie slov Pogamut, Java a ACT-R

rozšírenia PoJACT-R. V závere sú zhrnuté výsledky práce a možná budúca práca na projekte.

# Kapitola 2

# Kognitívna architektúra ACT-R

ACT-R je teória o modelovaní ľudskej mysle, ktorej korene siahajú 30 rokov do minulosti[1]. Za ten čas sa vyvinula v komplexnú teóriu, ktorá skúma rôzne aspekty ľudskej kognície. Táto kapitola vysvetľuje základy ACT-R a oblastí z ktorých čerpá.

## 2.1 Jemný úvod do problematiky

Teória ACT-R vychádza z viacerých odborov, ako je napríklad odbor umelej inteligencie, kognitívnej psychológie a neurovedy. Táto kapitola načrtne niektoré oblasti potrebné pre pochopenie základov ACT-R.

### 2.1.1 Kognitívna psychológia

Kognitívna psychológia sa časom, podobne ako iné vedné disciplíny, rozdelila na poddisciplíny. Tieto skúmajú rôzne aspekty kognície, ako napríklad: pamäť, pozornosť, percepcia a myslenie. To podnecuje logickú otázku: *Ako jednotlivé časti pospájať dohromady?* ACT-R sa snaží ukázať akým spôsobom tieto špecializované teórie pospájať, aby vytvorili teóriu koherentnej mysle.

[1] Presnejšie o tom pojednáva Anderson v [4] s. 39-43.

### Pamäť

V psychológii je pamäť definovaná ako schopnosť organizmu (človeka) ukladať, zachovať a obnoviť objekty [3]. Pamäťe môžme rozdeliť podľa času uchovávania:

**Senzorická pamäť** je pamäť, kde sa uchovávajú objekty tesne po ich spozorovaní. Príkladom je schopnosť „vidieť“ krátku chvíľu aj pri zatvorení očí. Detaily obrazu sa však rýchlo strácajú. Odhadovaný čas uchovania informácie v sensorickej pamäti je 200-500 milisekúnd.

**Krátkodobá pamäť** dovoľuje uchovávať limitovaný počet objektov od desiatok sekúnd do minúty. Experimenty ukazujú, že kapacita krátkodobej pamäte je 4-5 objektov. Tieto objekty však nemusia byť jednoduché, len ich človek musí vedieť komprimovať na jeden objekt. Táto schopnosť sa nazýva chunkovanie a takto komprimované objekty chunky. Príkladom je zapamätanie si sekvencie písmen: náhodná sekvencia (napríklad KDGIEPSNH) je pre väčšinu ľudí prakticky nezapamätateľná, avšak sekvencia s nejakým vzorom (napríklad AAABBBCCC) je zapamätateľná triviálne (toto je aj dôvodom rozdeľovania telefónnych čísel na trojice).

**Dlhodobá pamäť** uchováva spomienky od niekoľkých minút po desiatky rokov. Jej kapacita sa zdá obrovská, pretože človek sa dokáže učiť nové veci po celý život.

**Deklaratívna pamäť** ukladá spomienky a fakty, ktoré sú vedome prístupné. Rozdeľuje sa na sémantickú a epizodickú.

**Epizodická pamäť** sa zaoberá spomienkami na konkrétne (autobiografické) epizódy ľudského života. Epizodické spomienky sú vždy asociované s konkrétnym časom, miestom a emočným alebo iným kontextom. Epizodická pamäť dokáže odpovedať na to, čo bolo včera na obed, alebo aká bola rodinná dovolenka.

**Sémantická pamäť** sa uchováva všeobecné znalosti o svete a vedomosti, ktoré sa človek naučí. Typickým príkladom sú fakty typu „Mačka je zviera“ alebo „PI=3.14“.

**Procedurálna pamäť** odpovedá schopnostiam a procedúram, ktoré sa človek za svoj život naučil. Príkladom môže byť bicyklovanie,

alebo hra na hudobný nástroj. Procedurálna pamäť ukladá vedomosti o tom, ako úspešne vykonať nejakú úlohu.

Existujú aj iné delenia, ktoré odrážajú špecializáciu kognitívnych výskumníkov na určitý aspekt dlhodobej pamäte. Za zmienku stojí emocionálna pamäť, ktorá ukladá udalosti so silným emocionálnym kontextom. Emoionálna pamäť môže hrať veľkú rolu v procesoch, ktoré riadia deklaratívnu a procedurálnu pamäť. ACT-R sa zatiaľ touto pamäťou nezaoberá, avšak pre plausibilné modelovanie ľudského správania je veľmi dôležitá.

### Učenie

Rôzne spoločenstvá toho istého druhu živočíchov (špeciálne to platí pre človeka) žijú v rozličnom prostredí. Aby to dokázali, evolúcia ich na to pripravila viacerými formami učenia dlhodobej pamäte. Vďaka nej sa počas života dokážu prispôsobiť konkrétnemu prostrediu. Učenie môžme rozdeliť na niekoľko druhov:

**Učenie nových faktov:** Človek dokáže ukladať do svojej deklaratívnej pamäťe nové objekty. Či už spomienky do epizodickej pamäte, alebo fakty do sémantickej pamäte.

**Posilňovanie:** Pri opakovanom obnovovaní faktu (alebo spomienky) sa fakt posilní a je pravdepodobné, že bude zabudnutý neskôr.

**Získavanie schopností:** Ďalším druhom učenia je učenie nových procedúr. Jednoduchým príkladom je písanie na klávesnici. Postupným trénovaním sa človek naučí písať automaticky bez pozerania na klávesy. Zároveň však môže mať problémy s vedomým nájdením písmena, pretože túto procedúru už nepoužíva.

**Podmieňovanie:** Naučené procedúry môžu byť rôzne efektívne v závislosti na kontexte v ktorom sú vykonávané, preto sa vytvárajú podmienky na ich spustenie. Toto učenie môžme dalej deliť na:

**Klasické podmieňovanie:** Prvý ho skúmal Pavlov pri jeho experimentoch s podmienenými reflexmi psov. V tomto type je správanie podmieňované nadradenou udalosťou.

**Operačné podmieňovanie:** Narozdiel od klasického sa podmieňuje prostredím a dôsledkami danej procedúry. Atraktívnosť danej procedúry závisí od toho ako v minulosti dopadlo jej vykonávanie, teda či boli dôsledky príjemné pre vykonávateľa.

### 2.1.2 Kognitívne architektúry

Kognitívna architektúra (ďalej KA) je pojem, ktorý je často používaný, ale ťažko definovateľný. KA vo všeobecnosti navrhuje výpočtové procesy, ktoré sa správajú ako inteligentný systém. Najčastejšie ako osoba—agent, prípadne inteligentne podľa inej definície. Pojem architektúra implikuje prístup, ktorý sa pokúša modelovať nielen výsledné správanie systému, ale aj vnútornú organizáciu systému. Tento pojem tiež naznačuje, že KA sa zaoberá mapovaním štruktúry na funkčnosť. Štruktúrou rozumieme anatómiu centrálneho nervového systému (ďalej CNS) a jej modelov. Naopak funkciu tvoria behaviorálne aspekty ľudskej, prípadne živočíšnej inteligencie.

Predtým ako existoval pojem KA, výskumníci zaujímajúci sa o kogníciu mali len dve možnosti skúmania. Buď sa zaoberať štruktúrou a stratiť sa v komplexnosti CNS (odhaduje sa, že CNS človeka tvorí okolo 100 miliárd neurónov). Druhá možnosť je skúmať funkciu, a stratiť sa v detailoch ľudskej psychológie. Rozličné KA sa snažia ukázať, že pre pochopenie mysle potrebujeme pochopiť väzbu medzi štruktúrou a funkciou (medzi časťami CNS a vedomím) a nie skúmať každú oblasť zvlášť. Otázkou však zostáva, aká úroveň abstrakcie je najlepšia pre špecifikáciu ľudskej kognitívnej architektúry.

Pretože táto práca sa zaoberá teóriou ACT-R použijeme definíciu jej autora, John R. Andersona [4] :

> Kognitívna architektúra je špecifikácia štruktúry mozgu takej abstrakcie, aby vysvetlovala ako dosahuje funcie mysle.

Rôzne KA sa zaoberajú rôznymi aspektami ľudskej mysle. V začiatkoch umelej inteligencie sa zaoberali najmä riešením problémov, plánovaním a učením. V posledných rokoch KA začínajú počítať s väčšou komplexnosťou mysle a zaoberajú sa aj pozornosťou, výberom akcií a emocionálnymi modalitami (motiváciou, postojmi a emóciami agentov).

### 2.1.3 Produkčné systémy

Produkčný systém (Production system, Production rules system) je skupina počítačových systémov používaná v umelej inteligencii hlavne na riešenie problémov (problem solving). Produkčné systémy pozostávajú z troch častí: globálna databáza, produkčné pravidlá a kontrolný systém.

**Globálna databáza** je krátkodobá pamäť systému. Je to množina faktov, časť je stála (axiómy prostredia) a časť reprezentuje súčasný stav prostredia systému.

**Produkčné pravidlo** je dvojica podmienka-akcia. Kedykoľvek je podmienka v produkčnom systéme splnená, systém má dovolené vykonať danú akciu.

**Kontrolný systém** sa stará o konflikty pravidiel (ak sa ich dá aplikovať viac) a o poradie (ak je to dôležité pre vyriešenie problému).

Pre lepšiu ilustráciu nasleduje jednoduchý produkčný systém:

```
MP <= mnozina pravidiel
DATA <= aktualna globalna databaza
opakuj kym ciel nie je splneny{
    aktualizuj DATA
    vyber z MP podmozinu aplikovatelnych P
    ak P je neprazdna{
        vyber pravidlo X z pravidiel P
        vykonaj pravidlo X
    }
```

### 2.1.4 Symbolická verzus subsymbolická reprezentácia

Debata o tom, či je ľudská kognícia postavená na symbolickej alebo subsymbolickej reprezentácií pokračuje už vyše 30 rokov bez známok rozhodnutia [4], [7]. V princípe sa jedná o to, s čím pracuje ľudský mozog.

- Používa symboly a väzby medzi nimi ako reprezentáciu znalostí a následne ich spracováva.
- Používa distribuovanú reprezentáciu znalostí a nejakým spôsobom túto reprezentáciu spracovava komplexným a zmysluplným spôsobom.

Symbolická reprezentácia sa dá jednoducho vysvetliť na metafore s počítačom: Čo počítač robí je, že spracováva symboly na vstupe na symboly na výstupe. Tieto symboly reprezentujú nejaký koncept z reálneho sveta. Počítač dokáže s týmito symbolmi manipulovať použitím sady inštrukcií. Ľudská kognícia podľa zástancov symbolickej teórie funguje podobným manipulačným procesom.

Subsymbolická (alebo konekcionistická) reprezentácia je asociovaná s metaforou, respektíve modelom neurónu. Rané implementácie subsymbolických systémov pozostávali z jednoduchého modelu neurónu—perceptronu. Ako skupina neurónov určitej funkcie aj subsymbolický systém je postavený na skupine perceptronov pospájaných navzájom spojmi s rôznymi váhami. Naučený subsymbolický systém je potom schopný rozoznať vstup a podľa váh vygenerovať výstup. Tieto váhy sa dajú učiť rôznymi (aj autonómnymi) algoritmami a to je jedna z najväčších výhod subsymbolických systémov.

Následné dva body ukazujú základné rozdiely medzi symbolickými a subsymbolickými systémami:

- Symbolické systémy pracujú sériovo zatiaľ čo subsymbolické pracujú paralelne.
- Subsymbolické systémy sa hlavne zaoberajú rozoznávaním vstupov a subsymolické skôr manipuláciou už rozoznaných symbolov.

Aj keď je tu dichotómia týchto dvoch prístupov, podľa [4] sa dá na nich nahliadať ako na dve strany tej istej mince. Konkrétnejšie: subsymbolické systémy rozoznávajú vstup a predávajú predspracované informácie do symbolických systémov.

### 2.1.5 Biologické pozadie a modularita mozgu

Človek žije v komplexnom prostredí, ktoré mu ponúka simultánne množstvo podnetov a zároveň od neho vyžaduje zmysluplnú odpoveď. Podľa [4] tieto informácie a požiadavky sa dajú nahrubo rozdeliť na:

**Percepčné (zmyslové):** ) Človek potrebuje spracovávať informácie z rôznych zdrojov a zároveň extrahovať z toho množstva len tie informácie, ktoré sú potrebné. Napríklad pri šoférovaní auta musí sledovať ostatné vozidlá a počúvať, či nezačuje zvuk sirény.

**Motorické (manuálne, svalové):** Človek potrebuje vedieť vykonávať rozličné úkony a často viac úkonov simultánne. Príkladom je ovládanie volantu a stláčaní pedálov pri šoférovaní.

**Kontrolné (koordinačné):** Ľudské vnemy, myšlienky a konania musia byť koordinované, aby dokázali uspokojiť ľudské potreby. Jedným z dôvodov je, že veľa činností pozostáva z množstva úkonov, ktoré musia byť usporiadané v správnom poradí. Príkladom je uvarenie večere.

Ak sa vynechajú potreby reči, ľudské motorické a percepčné schopnosti sú vo veľkej miere podobné ostatným primátom. Koordinácia je však rozdielna. Schopnosť vyvinúť nové formy správania dala ľudskému druhu unikátnu možnosť dosiahnuť vysokú efektivitu v širokej škále činností, na ktoré nebol implicitne pripravený evolúciou. Príklady takýchto komplikovaných činností zahŕňajú napríklad servírovanie čaju pri japonskom čajovom obrade, operácie srdcovej chlopne v modernej chirurgii alebo krasokorčuľovanie.

Tieto schopnosti sú o to zaujímavejšie, že mozog, akokoľvek je komplikovaný, má limitované prostriedky na ich vykonávanie. Ľudský nervový systém pozostáva zhruba zo 100 miliárd neurónov. Je to relatívne pomalý systém (pri porovnaní s modernými počítačmi), avšak enormne paralelizovaný. Zo štruktúry neurónov vyplývajú dve zásadné limitácie systému.

1. Každý neurón má svoju veľkosť a svoje biologické potreby, aby mohol správne vykonávať svoju funkciu. Evolúcia optimalizovala pomer medzi veľkosťou a spotrebou zdrojov (priestorových a energetických). Preto má ľudský mozog 100 miliárd a nie 100 biliónov neurónov.

2. Podobná limitácia súvisí s komunikáciou. Neuróny sú rozmiestnené po celom mozgu a platí, čím ďalej sú dva neuróny od seba vzdialené, tým dlhšie ich komunikácia trvá a tým viac energie a priestoru spoje vyžadujú. Toto pravidlo muselo počas evolúcie vyvolávať selekčný tlak na lokalizáciu spojov. Aj súčasné znalosti o objeme mozgu, spotrebe energie, počte neurónov a spojov ukazujú, že väčšina komunikácie je lokálnej.

Druhý bod je základným faktom podporujúcim teóriu modularity mozgu. Tento fakt samotný však modularitu nedokazuje. Ďalším dôkazom by mohli byť experimenty s funkčnou magnetickou rezonanciou (fMRI), ktoré ukazujú,

že pri rôznych úlohach sú aktívnejšie ohraničené časti mozgu. Tieto časti sa dajú predstaviť ako moduly z určitou funkciou. Modulárny systém by mohol byť to k čomu evolúcia smerovala. Ak sa však zoberie do úvahy koľko sa toho o mozgu ešte nevie, stále je to veľmi odvážne tvrdenie.

### 2.1.6 Modulárna architektúra a Fodorove moduly

Ako názov naznačuje modulárna architektúra systému znamená, že je postavený z viacerých modulov plniacich rôzne funkcie. Fodor sa vo svojej knihe Modularity of mind [6] zastáva myšlienky modularity mozgu a navrhuje základné vlastnosti týchto modulov. Ako ďalšia kapitola ukáže, teória ACT-R je Fodorom veľmi ovplyvnená, aj keď s niektorými bodmi návrhu nesúhlasí. Niektoré Fodorove návrhy o moduloch mozgu ako ich zhrnul Anderson [4]:

**Špecificita domény:** Fodor tvrdil, že modul spracováva iba obmedzenú množinu jemu patriacich stimulov.

**Záväzné operácie:** Fodor si myslel, že keď konkrétny stimul príde do systému, moduly musia reagovať a ako reagujú nemôže byť zmenené. Preto človek nemôže zmeniť pohľad akým vníma svet.

**Zapuzdrenie informácie:** Fodor tvrdil, že väčšina informácii je v rámci modulu zapuzdrených a modul nepotrebuje posielať požiadavky na iné moduly.

**Rýchle operácie vrámci modulu:** Fodor si myslel, že ako priamym dôsledkom zapúzdrenia informácií je, že modulárne procesy sú najrýchlejšie kognitívne procesy

**Plytké výstupy:** Fodor vysvetľoval, že výstup jednotlivých modulov je plytký, teda že systém propaguje jednoduché a nie zložité fakty.

**Fixovaná neurónová štruktúra:** Fodor bol zástancom myšlienky, že každý modul má svoje dedikované neurónové štruktúry.

**Jazyk:** Fodor tvrdil, že existuje dedikovaný modul na spracovanie syntaxe reči.

**Obsahová-špecificita modulov:** Kognitívni výskumníci navrhli množstvo obsahovo špecializovaných modulov, ako napríklad modul na rozpoznávanie tvárí alebo modul na detekciu klamárov. Fodor nesúhlasil s takýmto druhom modulov.

**Centrálna kognícia:** Fodor sa špecializoval na vstupné a výstupné systémy. Tvrdil, že neexistuje centrálne spracovanie. Konkrétne tvrdil, že v mozgu neexistuje centrum pre logickú operáciu Modus Ponens.

## 2.2 Adaptive Control of Thought-Rational

Kapitola o ACT-R chce vysvetliť jeho základy. Nechce opísať celú teóriu, skôr ponúknuť náhľad, ktorý je potrebný pre pochopenie implementácie PoJACT-R. Kapitola čerpá hlavne z [4] a [5], ktoré obsahujú detailnejšie vysvetlenie niektorých častí ACT-R.

### 2.2.1 ACT-R základný slovník

**Chunk:** jednotka informácie, ktorej veľkosť je nastavená podľa princípov kognitívnej psychológie. V zásade sa jedná o malé množstvo informácie s ktorým dokáže mozog pracovať ako s jedným objektom.

**Slot:** Chunk sa skladá z niekoľkých slotov, ktoré obsahujú jednoduchú informáciu. Napríklad slot ıfarba obsahuje dáta ıčervená.

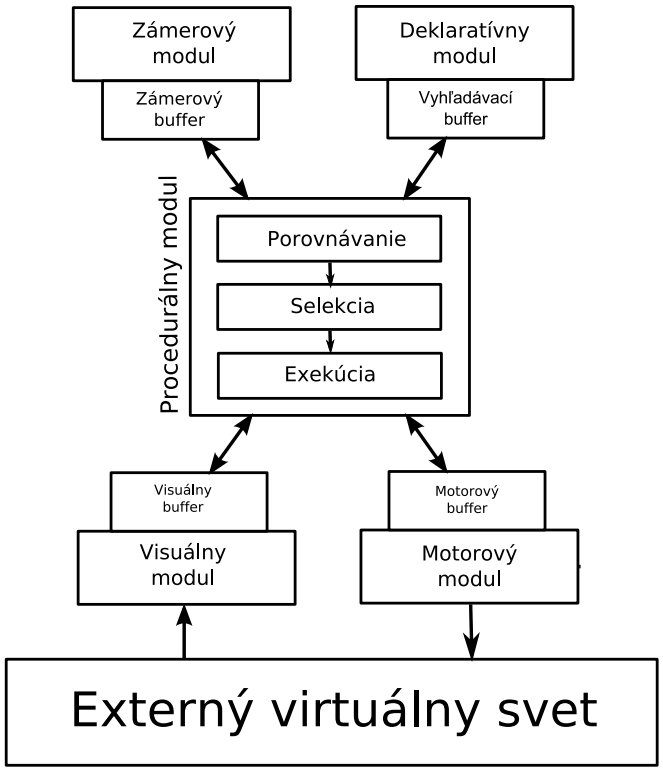
**Buffer:** zásobník chunkov

**Modul:** Komponenta, ktorá sa zaoberá spracovaním konkrétneho druhu informácií.

**Model:** Je komplex modulov a ich bufferov, ktorý sa snaží napodobiť nejaký aspekt ľudskej kognície.

### 2.2.2 Základ architektúry

ACT-R je teória navrhujúca modulárnu kognitívnu architektúru, niekoľko jej modulov a spôsob akým tieto moduly produkujú koherentnú kogníciu. ACT-R obsahuje napríklad percepčno-pohybové moduly, zámerový modul, deklaratívny modul. Každý z týchto modulov je asociovaný s inou oblasťou




Obr. 2.1: ACT-R architektúra

kortexu. Moduly majú asociované buffere, do ktorých vkládajú informácie v podobe chunkov. Na vzorce chunkov v bufferoch môže reagovať centrálny produkčný systém, ktorý v každom okamihu vyberie jedno produkčné pravidlo, ktoré sa použije. Subsymbolické procesy riadia vybratie vhodného pravidla pri nejednoznačnosti a taktiež ich používajú interne niektoré z modulov.

Obrázok 2.1 popisuje architektúru ACT-R. Tá pozostáva zo skupiny modulov, z ktorých každý je zameraný na spracovanie rôznych druhov informácií. Zámerový modul (Goal modul) udržiava súčasný cieľ, vizuálny modul (Visual modul) spracováva informácie z vizuálneho poľa, motorický modul (Motor modul) kontroluje pohyb a deklaratívny modul (Declarative modul) uchováva a získava informácie z dlhodobej pamäte. Každý modul má asociovaný buffer (prípadne zopár bufferov), v ktorých ponúka limitovaný objem informácií (niekoľko chunkov).

Moduly navzájom nekomunikujú, o koordináciu týchto modulov sa skrz buffere stará procedurálny modul[2]. Tento modul však okrem informácií v bufferoch nie je senzitívny na procesy v ostatných modulov. Buffere tvoria jedno z úzkych hrdiel teórie, ktoré má opodstatnenie v reálnej ľudskej limitácii stimulov.

Typickým príkladom je ľudská vizuálna pozornosť: človek si nie je vedomý celého vizuálneho poľa, ale len tej časti, na ktorú je upriamená jeho pozornosť (tá časť sa potom dostane do bufferu). Ak však je nejaký stimul zaujímavejší, respektíve evolučne dôležitejší (napríklad padajúci strom) pozornosť sa upriami naň a dostane sa do bufferu. Tam naň môže reagovať procedurálny modul a propagovať informáciu ďalej (napríklad do buffera manuálneho modulu sa pošle príkaz na vyhnutie sa).

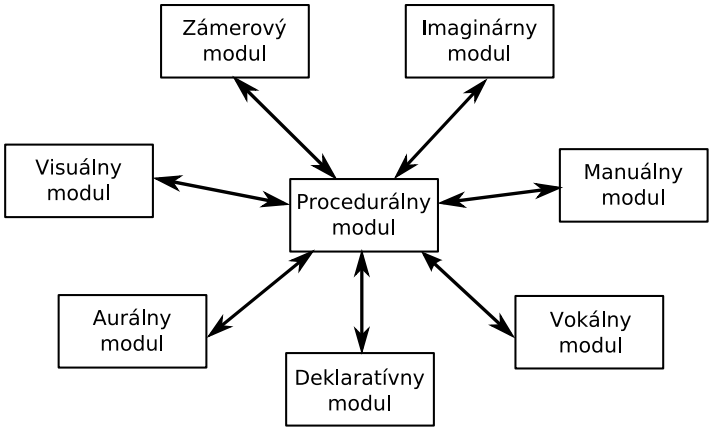
### 2.2.3 Moduly ACT-R

Obrázok 2.2 ilustruje 8 modulov v súčasnosti zahrnutých do teórie ACT-R. Patria tam dva percepčné moduly: vizuálny spracujúci vizuálne pole a aurálny spracujúci zvuky. Dva motorické moduly: manuálny ovláda svaly tela a vokálny ovláda hlasivky. Zvyšné 4 moduly sú zodpovedné za rôzne aspekty centrálneho spracovania. Sú to: zámerový modul, imaginárny modul, deklaratívny modul a procedurálny modul. Každý z týchto modulov je principiálne schopný masívne paralelného spracovania. Napríklad vizuálny modul paralelne sleduje a vyhodnocuje celé vizuálne pole a deklaratívny dokáže prehľadávať veľkú databázu spomienok.

#### Procedurálny modul

Procedurálny modul tvorí kognitívne jadro systému ACT-R. Je to most medzi stimulmi zmyslového vnímania (napr. zraku), interných vedomých informácií (napr. zámeru) a reakciami človeka (napr. ovládanie hlasiviek). Jednotlivé prechody tohto mostu sú uložené v procedurálnej pamäti ako procedurálne pravidlá. Produkčný modul sa zjednodušene dá predstaviť ako

[2]V staršej verzii ACT-R 5.0 sa tento modul volá Centrálny procesný systém (Central processing system)




Obr. 2.2: Moduly ACT-R

produkčný systém, ktorého globálnu databázu tvorí aktuálny obsah bufferov a množinu pravidiel tvorí procedurálna pamäť. Táto analógia je dôležitá, avšak treba mať na pamäti, že procedurálny modul má biologické pozadie, ktoré túto analógiu podporuje. Je to zároveň trochu neštastná analógia, ktorá je však bežným pochopením procedurálneho modulu. ACT-R nie je iba produkčným systémom ako ukazujú ostatné moduly a subsymbolická časť ACT-R[3].

Tak ako procedurálny systém aj procedurálny modul pracuje s produkčnými pravidlami. Tie v ACT-R tvorí trojica: podmienka, akcia a utilita. Podmienka tvorí šablónu pre buffere a určuje či je pravidlo použiteľné pri danom naplnení bufferov. Akciu tvoria modifikácie bufferov a ich chunkov. Utilita je modalita, ktorá hovorí aká je účinnosť tohoto pravidla pri splňovaní aktuálneho zámeru.

Kým ostatné moduly pracujú nezávisle, a ich výkon nič neobmedzuje, kritickou limitáciou je práve cyklus procedurálneho modulu. Tento cyklus pozostáva z troch fáz:

[3]V [1],[2] sú detailne popísane mozgové štruktúry, na ktoré sú namapované jednodlivé moduly ACT-R. Toto mapovanie a výsledký fMRI sú zaujímavé, ale pre rozsah sa nimi táto práca nezaoberá.

**Podmieňovanie:** V tejto fáze procedurálny modul prehľadáva procedurálnu pamäť podľa obsahu bufferov a zisťuje, ktoré produkčné pravidlá sú aplikovateľné.

**Selekcia:** Porovnávanie mohlo označiť viac aplikovateľných pravidiel. Vyberá vždy to s najväčšou utilitou. Spôsob vypočítavania utility je načrtnutý v podkapitole 2.2.5.

**Exekúcia:** Nakoniec sa vybrané produkčné pravidlo vykoná a buffere sú modifikované pre ďalší cyklus.

ACT-R predpokladá, že tento kognitívny cyklus trvá približne 50ms.

### Percepčno-pohybové moduly

ACT-R sa vo svojich začiatkoch zaoberal vyššími formami kognície, nie percepciou a ovládaním tela. Avšak senzoricko-motorický systém je rovnako komplexný a dôležitý ako centrálny systém, pretože dokáže veľa napovedať o vstupoch a výstupoch procedurálneho modulu. ACT-R obsahuje teóriu o motorických, vokálnych, aurálnych a vizuálnych moduloch, ktorými sa pre rozsah táto práca nezaoberá.

### Zámerový modul

Jedným z výhod ľudského druhu je schopnosť abstrakcie prijatých informácií a rozmýšlanie v symboloch. Ak človek uvidí dva objekty (napríklad čísla), ponúkne sa mu niekoľko možností ako na tieto objekty reagovať. Podľa toho aký je jeho súčasný zámer si vyberie, a to bez explicitných vonkajších podnetov. Tento zámer môže byť súčasťou alebo dôsledkom iného *vyššieho* zámeru (napríklad pred varením je potrebné ísť do obchodu nakúpiť suroviny).

Zámerový modul udržiava stopu týchto zámerov, aby správanie zodpovedalo súčasnému zámeru.

### Imaginárny modul

Pri riešení problémov potrebujú ľudia vedieť udržať nielen zámer, ale krátkodobo aj aktuálny stav vykonávania (napríklad pri výpočte rovnice udržať

aktuálny medzivýsledok). Túto krátkodobú pamäť v ACT-R zastupuje imaginárny modul, respektíve jeho buffer.

#### Deklaratívny modul

Deklaratívna pamäť je modul, vďaka ktorému si agent dokáže pamätať fakty a spomienky. Ľudská deklaratívna pamäť je obrovský sklad informácií, schopný masívneho paralelizmu pri získavaní informácií v ňom uložených. Jednoduchým príkladom je fakt *Lincoln bol prezidentom spojených štátov*. Napriek jednoduchosti informácie, prístup k nej nie je vôbec bezproblémový. Teória ACT-R sa vo veľkej miere zaoberá práve subsymbolickými aktivačnými procesmi, ktorými je potom deklaratívna pamäť riadená.

### 2.2.4 Symbolický versus subsymbolický ACT-R

ACT-R je hybridná kognitívna architektúra, teda má symbolickú aj subsymbolickú úroveň. Symbolická úroveň v ACT-R je abstraktná charakterizácia reprezentácia znalostí v mozgu. Subsymbolická úroveň je abstraktnou charakterizáciou neurónových procesov pri spracovaní znalostí.

Zjednodušene povedané symboly ponúkajú distálny prístup nutný k presunu informácií z jedneho miesta na druhé. Subsymbolická úroveň hovorí konkrétne, ktorá informácia sa bude prenášať a aká bude rýchlosť prenosu.

Teória ACT-R sa vo väčšej miere zaoberá symbolicko-subsymbolickým odlišnostiam v dvoch moduloch: deklaratívnom a procedurálnom.

#### Symbolicko-subsymbolické odlišnosti v deklaratívnom module

Deklaratívny modul používa sieť znalostí kódovaných v chunkoch. Pri požiadavke na deklaratívny modul sa musí vrátiť odpovedajúci chunk. Ak ten chunk neexistuje, alebo existuje viac odpovedajúcich chunkov ,je potrebné vybrať ten vhodnejší. Na to slúži subsymbolická úrovneň v podobe aktivácii chunkov. Najaktívnejší chunk bude vrátený a na veľkosti aktivácie závisí kedy bude vrátený. Aktivácie sú počítane tak, aby odrážali dôsledky Hebbovského učenia a propagácie aktivácie cez neuróny.

**Symbolicko-subsymbolické odlišnosti v procedurálnom module**

Subsymbolické procesy v procedurálnom module sa zaoberajú kompiláciou nových pravidiel a učením utitily.

Učenie nových pravidiel nastáva vtedy, ak sa model dostane do situácie, pre ktorú nemá explicitne definované produkčné pravidlá. Učenie v ACT-R sa spolieha na to, že má model definované základné všeobecné pravidlá. Na nich potom stavia komplikovanejšie správania, alebo zlešuje ich vykonávanie. Typickým príkladom je dvojica pravidiel, kde prvé posiela požiadavok na deklaratívny modul, a druhé reaguje na výstup deklaratívneho modulu. Produkčný modul v tomto prípade vytvorí nové produkčné pravidlo, ktoré odstráňuje požiadavku na deklaratívny modul.

Utilita je modalita dôležita v druhom kroku kognitívneho cyklu, pretože pri konfliktných produkčných pravidlách sa vyberá to, ktoré ju má najväčšiu. Pri výpočte sa však počíta aj zo šumom, teda model môže reagovať na rovnakú situáciu rozdielnym správaním. Utility jednotlivých pravidiel sú nastavené podľa odmien, ktoré organizmus dostáva. Odmena znamená splnenie určitého zámeru v zámerovnom module. Keďže je problémom, že odmena sa môže dostaviť neskôr ako bolo pravidlo vykonané, odmeňujú sa všetky pravidlá od poslednej odmeny. Rozdielne sú však hodnoty odmien, ktoré klesajú s dĺžkou času prejdeného od ich exekúcie do odmeny.

Pri kompilácií produkčných pravidiel sa nastavuje utilita na 0, teda je len malá šanca, že sa použije. Produkčné pravidlo však môže byť vytvárané znovu a znovu, pri čom sa utilita zvyšuje a je väčšia šanca, že sa použijú. Navyše ak je to produkčné pravidlo úspešné, teda zrýchľuje dosiahnutie zámeru, bude dostávať väčšiu odmenu a jeho utilita vystúpi nad staré produkčné pravidlo.

## 2.3 jACT-R

K ACT-R teórii patrí simulátor, ktorý je voľne dostupný a bol vyvinutý práve pre potreby ACT-R teórie a experimentovanie s jej modelmi. Od svojho počiatku sa pre simulátor používal jazyk Lisp (Konkrétne ACT-R 6.0

používa implementáciu Common Lisp[4]). Pogamut je však imlementovaný v Jave, preto bolo treba buď naimplementovať most medzi platformami Java a CommonLisp, alebo nájsť iný, ľahšie pripojiteľný simulátor ACT-R. Víťazom (a nakoniec aj jediným kandidátom) sa stal jACT-R[5] (Java ACT-R), ktorý je priamo implementovaný na platforme Java ako rozšírenie pre Eclipse IDE. Jeho autorom je Anthony M. Harrison a heslom *Making cognitive science portable.*

### 2.3.1 Výhody a nevýhody jACT-R

Výhody a nevýhody rozhodnutia použiť jACT-R by sa dali zhrnúť do týchto bodov:

- Týmto výberom odpadla nutnosť komplikovaného prepájania Javy a Lispu.
- Pribudla možnosť písania modelov v modernom XML formáte (okrem štandardnej Lisp syntaxe)[6].
- Implementované moduly pre PoJACTR sú typovo bezpečné.
- Keďže jACTR je nový systém, nikým nefinancovaný, nie je ideálne zdokumentovaný a spočiatku mal veľa chýb.

## 2.4 Zhrnutie

Kapitola 2 vysvetlila základy ACT-R, potrebné pre pochopenie PoJACT-R. Zďaleka však nevysvetľuje všetky detaily a myšlienky z ktorých ACT-R vychádza. Ďalšie informácie sa dajú nájsť v [4], [5].

[4] http://common-lisp.net/
[5] http://www.jactr.org/
[6] Popis preferovanej XML syntaxe sa dá nájsť v užívateľskej dokumentácii PoJACT-R

# Kapitola 3

# Pogamut

Pochopenie princípov Pogamutu hrá kľúčovú rolu pre PoJACT-R, preto táto kapitola ukáže architektúru a niektoré návrhové vzory, ktoré Pogamut používa.

Pogamut[1] je projekt zameraný na vývoj virtuálnych bytostí—agentov. Snaží sa uľahčiť prácu výskumníkom umelej inteligencie a čo najviac ich oslobodiť od implementačných detailov virtuálnych svetov, aby sa mohli venovať výlučne UI. Pogamut je platforma navrhnutá pre zjednodušenie programovania a ladenia agentov. Základom zjednodušenia je vývojové prostredie (Integrated developement editor, alebo IDE). Pogamut používa Netbeans IDE[2] a jeho súčasťou je zásuvný modul (plugin) pre toto IDE. IDE dokáže vytvoriť agenta vo virtuálnom prostredí, kontrolovať ho a dovoľuje ladenie, zobrazenie a zmenu parametrov za behu. Platforma tiež obsahuje knižnicu a nadstavby, ktorá obsahuje základné riešenia niektorých problémov umelej inteligencie.

Pogamut, jeho rozšírenia ako aj plugin do Netbeans sú postavené na platforme a programovacom jazyku Java.

[1] http://artemis.ms.mff.cuni.cz/pogamut
[2] http://www.netbeans.org/

## 3.1 Unreal Tournament 2004 a Gamebots

Unreal Tournament 2004[3] (UT2004) je akčná hra typu FPS od vývojarského štúdia Epic Games. Je postavená na grafickej platforme Unreal Engine 2, ktorú používajú aj iné hry ako napríklad American Army. K UT2004 patrí editor máp UnrealEd a možnosť písania a púšťania skriptov v jazyku UnrealScript, čo ho povyšuje svet UT2004 na virtuálny svet použiteľný v iných odvetviach.

Pre Pogamut bol vyvinutý modul GameBots2004 (GB), ktorý je priamim potomkom modulu, ktorý vyvinuli Andrew N. Marshal a Gal Kaminka na University of Southern California's Information Sciences Institute. GB je napísaná v UnrealScripte, funguje ako zásuvný modul do UT2004 a jeho hlavnou úlohou je sprostredkovať komunikačný kanál medzi Pogamutom a UT2004. GB je štandardne súčasťou inštalačného balíka Pogamutu.

## 3.2 Pogamut verzie 2

Pogamut verzie 2 (ďalej P2) používa ako virtuálne prostredie Unreal Tournament 2004. Ako most medzi prostredím používa rozhranie GameBots. P2 obsahuje napríklad A* algoritmus pre hľadanie a plánovanie cesty a správu pamäte agenta. Sila P2, ale tkvie v jeho rozšíreniach. Tými sú napríklad pripojenie plánovača POSH, emočného modulu, knižnice pre genetické algoritmy a modul GRID zjednodušujúci paralelné púštanie experimentov. Najväčšou nevýhodou P2 je jeho tesná väzba s prostredím UT2004 a Netbeans IDE. Prvá väzba obmedzuje pripojenie iných virtuálnych svetov a druhá obmedzuje použitie P2 bez Netbeans IDE.

## 3.3 Pogamut verzie 3

V čase písania bakalárskej práce bola oficiálne vydaná verzia 2, avšak verzia 3 mala všetky kritické časti už implementované. Pogamut 3 (P3) by mal odstrániť nedostatky predchádzajúcej verzie. Výhody P3 sa dajú zhrnúť do nasledujúcich bodov:

[3] http://unrealtournament2004.com

1. Hlavnou motiváciu pre použitie P3 pre PoJACT-R je jeho udalosťami riadená architektúra, ktorá ako na mieru sedí pre použitý ACT-R simulátor.

2. P3 oddelené jadro a UT2004 závislé časti. Preto sa P3 bude dať napojiť aj na iné virtuálne svety.

3. P3 bude využívať plné výhody Java rozhraní, všetko je typovo bezpečné.

4. P3 využíva technológiu JMX podporujúcu možnosť ladenia Java kódu za behu agenta pri spustení z Netbeans Pogamut Pluginu.

## 3.4 Udalosťami riadená architektúra

P3 komunikuje s UT2004 pomocou GB a TCP/IP protokolu. Komunikácia funguje dvoma smermi:

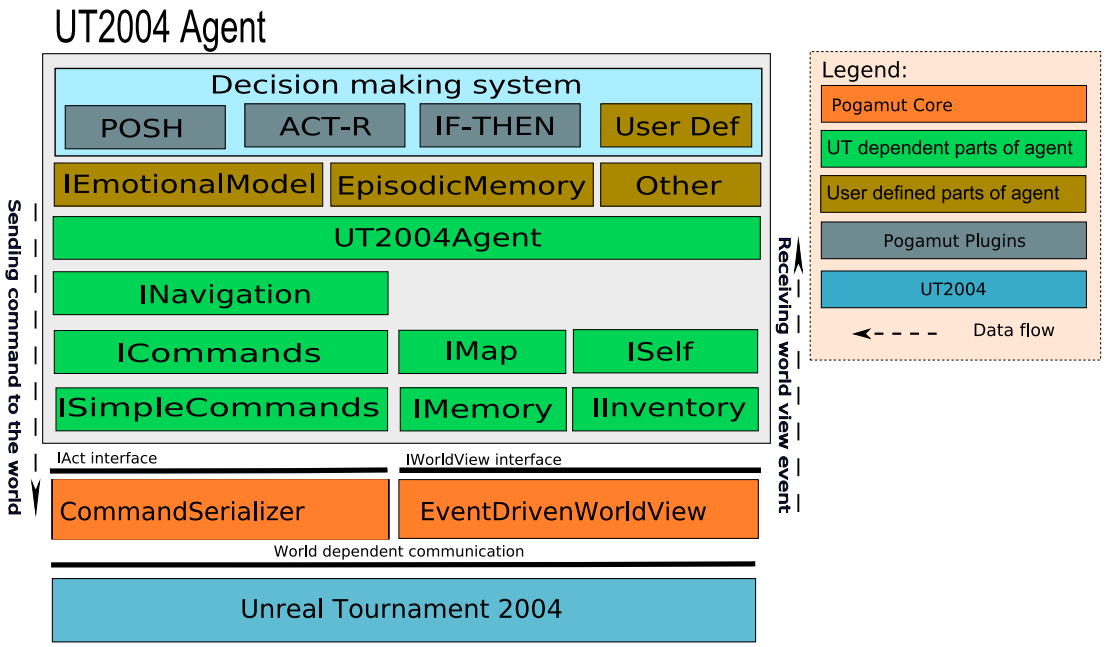
**UT2004 -> GB -> TCP/IP -> Pogamut:** UT2004 posiela správy o udalostiach v simulátore do GB, ten ich spracuje, serializuje spolu so svojimi správami a pošle cez protokol TCP/IP. Parser na strane Pogamutu ich dekóduje do Java objektov a posunie do `IWorldView` objektu. Tam už s nimi môžu pracovať moduly Pogamutu a jeho nadstavieb.

**Pogamut -> TCP/IP -> GB -> UT2004:** Ak chce Pogamut poslať príkaz do UT2004, inštanciuje zodpovedajúcu Java triedu príkazu. Tú predá cez `IAct` objekt do jadra Pogamutu, kde prebehne serializácia. Serializovaný objekt sa cez TCP/IP pošle do GB, tam sa deserializuje a vykoná. Či už interne GB, alebo sa príkaz spropaguje do UT2004.

### 3.4.1 Riadenie agenta

V P2 jediné miesto kde sa dalo vykonávať riadenie agenta bolo na metóde `IAgent.doLogic()`. Táto metóda sa púšťala dookola a medzi jednotlivými spusteniami sa vybavovala komunikácia. P3 má túto metódu tiež, avšak môže bežať vo vlastnom vlákne a nevynucuje sa jej použitie. Stačí registrovať listener na IWorldView a pracovať priamo s udalosťami.

Príklad na jednoduchú `IAgent.doLogic()` a na posielanie príkazu v P3:




Obr. 3.1: Unreal tournament 2004 agent v Pogamut 3

```
@Override
protected void doLogic() throws PogamutException {
  getAct().act(new TurnTo()
    .setRotation(new Rotation(50,50,50)));
}
```

Príklad na jednoduchý listener:

```
WorldEventListener<HearNoise> listener =
  new WorldEventListener<HearNoise>(){
    @Override
public void notify(HearNoise event) {
      getLogger().user().info("Hearing noise: " + event);
    }
  };

@Override
protected void prePrepareBot() {
  getWorldView().addListener(HearNoise.class, listener);
}
```

### 3.4.2 Spustenie agenta

V čase písania bakalárskej práce nebolo ešte rozšírenie Netbeans IDE pre P3 hotové. Preto nie je možné spustiť model cez IDE, ale je nutné ručne ho nakonfigurovať. Presný popis sa dá nájsť v užívateľskej dokumentácii PoJACT-R.

### 3.4.3 Google Guice

Google Guice[4] je jednoduchý framework na vkladanie závislostí pre jazyk Java. Zjednodušene povedané Guice znižuje nutnosť používania *Factory* tried a zlepšuje možnosti zmeny existujúceho kódu, testovania a používania kódu v inom kontexte. V PoJACT-R sa používajú obidve paradigma.

[4]Lightweight dependency injection framework http://code.google.com/p/google-guice/

# Kapitola 4

# PoJACT-R

PoJACT-R spája dva robustné systémy Pogamut a jACT-R. Ako už bolo spomenuté v úvode, PoJACT-R musel vyriešiť 4 základné otázky:

1. Ako zachovať plausibilitu ACT-R
2. Ako zjednodušiť použitie pri zachovaní komplexnosti
3. Ako čo najviac využiť typovú bezpečnosť jazyka Java
4. Ako oddeliť jadro prepojenia a časťí závislých na UT2004

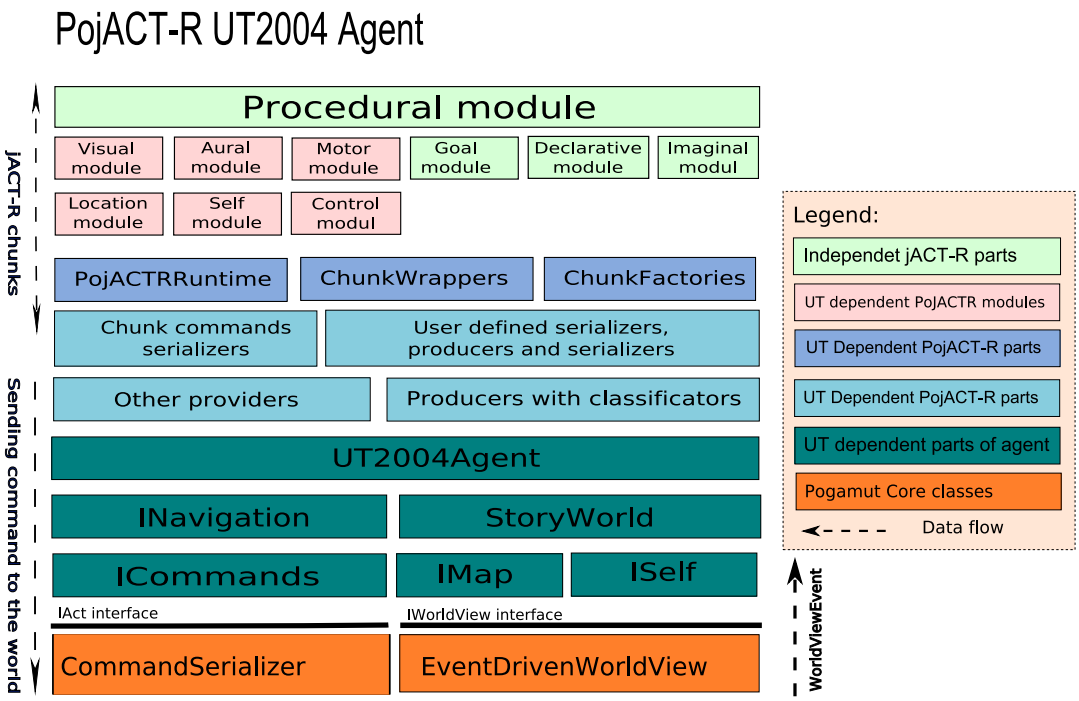
Počas riešenia týchto otázok sa objavili ďaľšie problémy, ktoré bolo nutné vyriešiť:

1. Synchronizácia vlákien
2. Preklad správ
3. Neúmerné množstvo logov

Ďaľšie podkapitoly sa venujú jednotlivým otázkam a riešeniam podrobnejšie. Sú zoradené podľa poradia v akom ich bolo nutné riešiť.

## 4.1 Architektúra PoJACT-R

Pre porovnanie s architektúrov Pogamutu na 2.2 je na obrázku 4.1 načrntutá architektúra PoJACT-R agenta. Z obrázku vidno, ktoré časti bolo nutné navrhnúť a implementovať pre PoJACT-R.




Obr. 4.1: PoJACT-R agent pre UT2004

## 4.2 Oddelenie jadra a častí UT2004

P3 bol navrhnutý tak, aby uľahčoval pripojenie ľubovolného virtuálneho sveta. To sa dosiahlo rozdelením Pogamutu na dva projekty:

**Core** obsahuje všetky triedy a funkčnosti, ktoré nezávisia na konkrétom svete.

**PogamutUT2004** obsahuje špecifické UT2004 triedy.

PoJACT-R by ako jeho rozšírenie malo túto vlastnosť zachovávať a využíva na to rovnaký spôsob. PoJACT-R je tiež rozdelený dvoch podprojektov:

**PoJACTRCore** obsahuje jadro rozšírenia a je závislé len Core projekte Pogamutu a jeho prerekvizitách. Obsahuje napríklad runtime triedy, všeobecných agentov, abstraktné triedy ACT-R modulov a všeobecné triedy chunkov.

**PoJACTRUT2004** obsahuje UT2004 závislé komponenty ako sú percepčno-motorické moduly, a defaultného UT2004 agenta.

Toto rozdelenie potom indukuje štandardný postup pri implementácii triedy závislej na simulátore. Trieda sa rodelí na: abstraktnú časť patriacu do jadra a konkrétnu UT2004 implementáciu rozširujúcu túto abstraktnú časť. Príkladom je `DefaultUT2004AuralProducer` patrí do PoJACTRUT2004 pričom dedí od abstraktnej triedy `AbstractAuralProducer` z jadra.

## 4.3 Typová bezpečnosť

P3 aj jACT-R využívajú plné výhody typovosti a generických tried ponúkaných jazykom Java a PoJACT-R, ako ich prepojenie by mal typovú kontrolu zachovávať. Pri implementácii bolo tomu venované veľa času, a do akej miery PoJACT-R rieši túto otázku je na posúdení čitateľa, alebo užívateľa. Treba však podotknúť, že na niektorých miestach sa muselo voliť medzi pretypovaním, alebo zásahom do zdrojových kódov jACT-R a ako menšie zlo bolo zvolené pretypovanie. Dôvodom bolo zachovanie jednoduchej aktualizácie jACT-R.

### 4.3.1 Trieda ChunkWrapper

Jednotkou informácie v ACT-R je chunk, ktorého symbolická časť sa skladá z dvojíc kľúč-hodnota. Simulátor jACT-R používa jednu všeobecnú triedu pre všetky chunky a typovosť je zaručená programovo. To však znamená, že programátor si musí kľúče jednotlivých slotov pamätať. Preto PoJACT-R používa obaľovaciu triedu, ktorá drží chunk a ponúka metódy pre prácu so slotmi. Príkladom je napríklad `AuralChunkWrapper`, ktorý obaľuje chunk v aurálnom module.

## 4.4 Zachovanie plausibility ACT-R

UT2004 je umelý svet a možnosti Pogamutu, čo sa týka komunikácie s týmto svetom, obmedzuje GB2004 a Unreal Script. To ide proti psychologicko-neurologickej plausibilite, o ktorú sa pokúša teória ACT-R.

Jedným z obmedzení je, že GB2004 štandardne posielaju správy o stave systému každých 250ms. Tento čas je síce konfigurovateľný, ale autori GB2004 ho nedoporučujú nastavovať pod 150ms kvôli stabilite systému. Keďže kognitívny cyklus ACT-R je 50ms, tak nie je možné dodávať simulátoru jACT-R aktuálne informácie. Tento problém sa snaží kompenzovať PoJACT-R buffer.

### 4.4.1 Aktivovaný chunk a jeho buffer

Pre chunky, ktoré používajú percepčné moduly PoJACT-R bol navrhnutý `ActivationChunkWrapper`. Rozširuje `ChunkWrapper` o aktiváciu, ktorá hovorí ako veľmi bol daný vnem zaujímavý (akým spôsobom sú chunky generované zo správ UT2004 je v kapitole 4.4.3). Aktivácia časom klesá podľa konfigurovateľnej funkcie. Sensorické buffere (napríklad `AbstractAuralBuffer` ) potom dedia od bufferu `AbstractPoJACTRActivationBuffer`, ktorý využíva aktiváciu následným spôsobom:

**Propagáciu implicitne dôležitých vnemov:** Vnemy v podobe chunkov sú zoradené podľa aktivácie a skrz buffer do centrálneho produkčného systému sú propagované len chunky s najväčšou aktivitou.

**Inhibíciu použitých vnemov:** Ak centrálny produkčný systém použije pravidlo, ktoré vychádzalo z istého chunku istého buffera, aktivácia tohto chunku je inhibovaná a iné vnemy dostanú priestor v ďaľšom kognitívnom cykle

**Excitáciu potrebných vnemov:** Niekedy centrálny produkčný systém potrebuje vedieť, či bol prijatý nejaký vnem (napríklad či je nepriateľ vo vizuálnom poli). Ak bol daný vnem v poslednom čase prijatý, jeho aktivácia sa zvýši a v daľšom cykle sa dostane do buffera. Ak daný vnem nebol prijatý, aktivuje sa chybový chunk, ktorý indikuje neexistenciu vnemu v buffery.

Treba podotknúť, že tento spôsob nemusí byť správnym plausibilným riešením, ani nie je podložený žiadnymi vedeckými prácami. Snaží sa však demonštrovať možnosti PoJACT-R a ukázať akým spôsobom je možné implementovať percepčné moduly. Vďaka modularite celého systému (kapitola 4.4.2), je možné jednotlivé moduly vymienať za nové. Pre výskumníkov zaujímajúcich sa o iné oblasti ako percepčné, by mal byť tento jednoduchý spôsob aktivácií postačujúci.

## 4.5 Komunikácia Pogamutu a jACT-R

Moduly a buffere jACT-R (a teda aj PoJACT-R) potrebujú byť instanciované počas inštalovania modelu a jACT-R nemá prístup k objektom Pogamutu a špeciálne PoJACT-R buffere potrebujú vidieť agenta ktorému patria.

### 4.5.1 PoJACTRRuntime

`PoJACTRRuntime` je singleton trieda, ktorá je všade viditeľná (a teda aj z jACT-R objektov) a riadi PoJACT-R. Mimoiného obsahuje aj metódy na inštalovanie modelov, na spustenie a zastavenie jACT-R. Tieto metódy sú volané z triedy `PoJACTRUT2004Bot` , od ktorej jACT-R boti pre UT2004 dedia. Dôležitou vlastnosťou `PoJACTRRuntime` je, že pri inštalovaní modelu vytvára mapovanie z agenta na model a naopak. To umožňuje PoJACT-R modulom a bufferom vyhľadať im patriaceho agenta podľa modelu ktorému patria.

### 4.5.2 Modularita spojenia

Pogamut využíva modulárny prístup knižnice Guice, ktorá dovoľuje výmenu rôznych súčastí Pogamutu (ako ukazuje kapitola 4.4). Táto vlastnosť je schovaná v triede `AgentWrapper` , ktorá obaľuje Pogamut agenta a dá sa nájsť na `PoJACTRRuntime` , teda je z jACT-R prístupná. `AgentWrapper` má metódu

```
public <K extends IPoJACTRModule> K getModule(Class<K> clazz);
```

v ktorej je ukryté použitie Guice injectoru. Vďaka tejto metóde PoJACT-R využíva modularitu knižnice Guice. `IPoJACTRModule` je markovacie rozhranie, ktoré slúži na identifikáciu vymeniteľných častí PoJACT-R.
K aktuálne používaným častiam patria takzvaný producenti, klasifikátory, serializéry a providery, ktorými sa zaoberá nasledujúca podkapitola.

### 4.5.3 Správy Pogamutu

Pogamut komunikuje s UT2004 pomocou správ dvoch druhov(ako presne je starosť Pogamutu, viz. podkapitola 3.4):

- Informatívne správy o stave sveta, ktoré posiela UT2004 Pogamutu.
- Príkazy, ktorými Pogamut riadi agenta a UT2004.

PoJACT-R prekladá tieto správy z a na chunky.

#### IChunkFactory

Typy chunkov s ktorými pracuje jACT-R musia byť registrované v deklaratívnom module. Tento typ je možné deklarovať modeli, alebo ručne v kóde. Pre zjednodušenie modelov sú typy chunkov, ktoré sú potrebné registrované v kóde pri inštalácii modelu. Podobne inštancie chunkov, ktoré sa používajú, musia byť pre jACT-R vytvorené deklaratívnym modulom.

Triedy, ktoré implementujú `IChunkFactory`, sú factory triedy schopné registrovať daný typ chunku, vytvárať pomocou deklaratívneho modulu nové chunky, alebo priamo ich obaliť do inštancie `ChunkWrapper`. Každú `IChunkFactory` triedu je potrebné nainštalovať do `PoJACTRRuntime`, kde sú potom voľne prístupné. Typickými konzumentmi týchto factory tried sú

producenti, ktorý sú predstavený v nasledujúcej podkapitole.

### Preklad správ na chunky

Mediátorom pre príchodzie správy sú takzvané producenti, ktoré dedia od `AbstractChunkEventProducer`. Práca producentov spočína v 4 úlohách

**Počúvanie správ:** `AbstractChunkProducer` vidí IWorldView, ktorý umožňuje registrovať na `IWorldView` listener na vybraný typ správ.

**Samotný preklad:** Po prijatí správy producent vygeneruje `ChunkWrapper` pomocou odpovedajúcej faktory triedy. Potom namapuje producent atribúty správy na jednotlivé sloty chunku (typovosť zabezpečuje obalenie v `ChunkWrapper`). Ak sa jedná o `ActivationChunkWrapper` môže producer aj nastavovať aktivitu podľa parametrov správy.

**Klasifikácia:** Nepovinnou, ale často využívanou vlastnosťou producenta je priradiť mu klasifikátor (dedí od `IChunkClassificator`). V takom prípade musí použitý `ChunkWrapper` implementovať rozhranie `IClassificable` , ktoré indikuje, že chunk má slot class. Klasifikátor podľa parametrov chunku klasifikuje chunk a priradenú triedu uloží do slotu class.

**Propagácie chunk eventu:** Keď je už dokončený preklad správy na chunk, vygenerovaný `ChunkWrapper` je obalený do triedy `ChunkEvent` a propagovaný všetkým listenerom registrovaným na producenta.

### Preklad chunkov na príkazy

Príkazom odpovedajú chunky typu `commandChunk` a pre nich určený `CommandChunkWrapper`. Tento chunk pozostáva zo slotu pre názov príkazu a slotov pre parametre, ktoré nie sú typované. Dôvodom na to bola veľká varieta príkazov pre UT2004 a pridanie vlastného typu chunku pre každý príkaz, by mohlo viesť k zbytočnému zneprehľadneniu kódu. Preto bol navrhnutý `AbstractCommandSerializer` , ktorý slúži na preklad príkazových chunkov a následné poslanie príkazov priamo do Pogamutu. Každý zo serializérov prekladá len niektoré príkazy a to tie, ktoré spolu logicky súvisia. Napríklad `MovementCommandSerializer` prekladá len tie príkazové chunky, ktoré súvisia s pohybom agenta.

#### Ostatná komunikácia

V niektorých prípadoch vyžšie uvedené spôsoby nestačia a sú potrebné špecifické mechanizmy. Vtedy je možné použiť providera. Provider je trieda, ktorá dedí od `AbstractProvider`. Jej účel a funkčnosť nie je nijako špecifikovaná. Príkladom je `AbstractLocationProvider`, ktorý dokáže podľa lokácie zistiť najbližší navigačný bod a používa na to `StoryWorld`, ktorý potrebuje špecifický prístup.

## 4.6 Jednoduchosť použitia pri zachovaní komplexnosti

Pogamut ako projekt má záujem plniť aj úlohu edukačnej platformy umelej inteligencie. Na strane druhej ACT-R je komplikovaná teória používaná vysoko školenými kognitívnymi výskumníkmi. PoJACT-R by preto mal byť použiteľný pri výuke umelej inteligencie a zároveň by si mal udržať možnosti komplikovaného ACT-R. Bolo navrhnutých niekoľko riešení, ktoré pre nekompletnosť P3 a nedostatok času nebolo implementovaných:

- Implementovanie pluginu do Netbeans IDE pre zjednodušenie spúštania modelov.
- Implementovanie editoru pre ACT-R modely, či už v syntaxy XML alebo Lisp.

## 4.7 Implementované ACT-R moduly

Pre ukážku PoJACT-R bol implementovaný model Hunter. Tento model pozostáva celkovo z X produkčných pravidiel a využíva niektoré doteraz implementované PoJACT-R moduly. Celý model Hunter možno nájsť v prílohe B. Implementované boli 4 percepčné a informatívne moduly.

#### Percepčné moduly

Aurálny modul (`DefaultAuralModule`) je percepčný modul, ktorý spracováva aurálne správy a propaguje ich do aurálneho buffera.

Vizuálny modul (`DefaultVisualModule`) je percepčný modul, ktorý spracováva visuálne správy a propaguje ich do dvoch bufferov, podľa typu správy. Jeden buffer obsahuje navigačné správy, ktoré definujú aktuálne viditeľné navigačné body. Druhým bufferom je objektový buffer, ktorý obsahuje aktuálne viditeľné objekty (ako je napríklad nepriateľský agent alebo lekárnička).

Lokačný modul (`DefaultLocationModule`) je modul, ktorý ma v buffery stále aktuálnu lokáciu agenta.

Vlastný modul (`DefaultSelfModule`) vo svojom buffery drží informácie o aktuálnom stave agenta, teda o počte životov, adrenalíne a brnení.

**Príkazové moduly**

Motorický modul (`DefaultBodyCommandModule`) ovláda pohyb a akcie agenta a má dva buffere. Pohybový buffer (`MovementCommandBuffer`) spracováva príkazy pre pohyb agenta v priestore. Telový buffer (`BodyCommandBuffer`) spracováva ostatné príkazy súvisiace s ovládaním agenta. Ako napríklad príkazy na streľbu, alebo odhodenie zbrane.

Kontrolný modul (`DefaultControlModule`) spracováva príkazy pre kontrolu servera a hry ako je napríklad zastavenie hry, alebo zmena týmu. Tento buffer nemá žiadne plausibilné opodstastnenie, napriek tomu je pre riadenie agenta potrebný.

## 4.8 Synchronizácia vlákien

P3 a jACT-R bežia vo vlastných vláknach a preto bolo nutné komunikáciu medzi nimi synchronizovať. Jazyk Java má synchronizáciu už vstavanú a jej použitie je jednoduché, avšak jACT-R používa vlastné synchronizačné primitíva. Oproti Java primitívam, ktoré automaticky zamykajú na zápis, jACT-R primitíva umožnujú zamykanie aj pre čítanie. Preto sa aj synchronizácia v PoJACT-R deje na úrovni jACT-R cez jeho primitíva.

## 4.9 Neúmerné množstvo logov

Cyklus v P3 trvá podľa nastavenia 100-300ms, cyklus v jACT-R trvá 50ms. Oba systémy generujú množstvo logovaných hlášok potrebných pre ladenie aplikácie. Niekedy je to však kotraproduktívne. Pri prvom spustení celého systému sa rýchlosť rastu logovacieho súboru pohybovala okolo 1MB/s. Po niekoľkých sekundách behu agenta sa prakticky z logu už nič nedalo vyčítať. Riešením tohoto problému bolo presmerovanie logov cez logovaciu knižnicu Log4j[1]. Tá umožnuje definovať filtre na jednotlivé logované hlášky. Tieto filtre sa nastavujú v konfiguračnom xml súbore a dajú sa pred každým spustením jednoducho editovať. To znamená, že implementátor modelu si môže nastaviť filtre tak, aby sa logovali len hlášky, ktoré sú potrebné pre jeho experimenty.

## 4.10 Zhrnutie

Dá sa povedať, že PoJACT-R vyriešil všetky kritické problémy a navrhol riešenia pre tie menej dôležité. Problémy si však vyžiadali daň v podobe rozsahu zdrojových kódov, ku ktorým je potrebné udržovať dokumentáciu. Na dokumentáciu a prehľadnosť kódu bol kladený veľký dôraz a preto si môže užívateľ jednoducho implementovať vlastný modul.

[1] http://logging.apache.org/log4j/1.2/index.html

# Kapitola 5

# Výsledky

Minulá kapitola ukázala ako PoJACT-R rieši, prípadne aké riešenia navrhuje pre svoje problémy. Nasledujúce podkapitoly hovoria o najjednoduchšom modely—lovcovi a ukazujú výkonnostnú štatistiku spojených systémov jACT-R a Pogamut pri použití tohto modelu. Treba podotknúť, že experimentov je málo, čo však nebolo spôsobené náročnosťou experimentovania, ale časovou náročnosťou implementácie samotného rozšírenia PoJACT-R.

## 5.1 Agent—lovec

Agent—lovec bol implementovaný ako jednoduchý príklad ACT-R modelu. Model obsahuje 11 modulov (z toho 6 PoJACT-R modulov), 10 bufferov (z toho 7 PoJACT-R bufferov) a 8 pravidiel s 1-2 podmienkami a 1-2 akciami. Lovcovo správanie je jednoduché: behá náhodne po mape, kým nezbadá nepriateľa. Keď ho zbadá, začne ho nahánať a strieľať po ňom. Naháňa a strieľa kým ho nestratí z dohľadu, alebo lovca nezabijú. Lovec je nasadený do malej mapy Training-Day.

Tento agent je veľmi jednoduchý a nerieši navigáciu, výmenu zbraní či zbieranie predmetov. Súbor s modelom je súčasťou štandardnej distribúcie PoJACT-R.

| Procesor | Intel(R) Core(TM)2 Duo E6750 2.66GHz |
|---|---|
| Chipset | Intel P31/P35 |
| Pamäť | 2x 1024MB DDR2-SDRAM |
| Grafická karta | NVIDIA GeForce 7600GS |
| Operačný systém | Microsoft Windows XP Professional SP2 |

Tabuľka 5.1: Parametre počítačovej zostavy použitej na testovanie

## 5.2 Výkon agenta—lovca

Pre testovanie bola použitá distribúcia jACT-R, Pogamutu a PoJACT-R s priloženého DVD. Parametre počítača sú uvedené v tabuľke 5.2.

Počas testovania boli okrem agenta zapnute programy: dedikovaný server UT2004, klient UT2004, IDE Eclipse Ganymede, AnVir Task Manager Pro. Agent bol spúšťaný z IDE Eclipse v ostrom móde (nie v ladiacom) a mal vypnuté logovanie, okrem zaznamenávania začiatku ACT-R cyklu.

Ako model pre testovanie bol použitý agent—lovec. Navrhnuté boli 2 experimenty:

- Skúmanie maximálneho počtu cyklov za minútu pri zrušení obmedzenia na 50ms cyklus ACT-R
- Skúmanie vyťaženia procesoru

Oba experimenty sa skúšali pri rôznom počte produkčných pravidiel, ktoré bolo dosiahnuté duplikovaním už existujúcich pravidiel v Lovec. Základný model Lovec má 8 pravidiel a testovalo sa 5,10 a 20 násobné zväčšenie počtu pravidiel. Odpovedajúce súbory sú

`hunter.jactr, hunterx5.jactr, hunterx10.jactr, hunter20.jactr`

a dajú sa nájsť v projekte PoJACTRUT2004/models.

### 5.2.1 Maximálny výkon

ACT-R teória predpokladá kognitívny cyklus dĺžky 50ms. To znamená, že ľubovolnému plauzibilnému virtuálnemu agentovi musí tento cyklus stačiť.

| Model | Pravidiel | Cyklov/min. | Priemerná dĺžka cyklu |
|---|---|---|---|
| hunter.jactr | 8 | 26041 | 2.3ms |
| hunterx5.jactr | 40 | 11156 | 5.3ms |
| hunterx10.jactr | 80 | 5606 | 10.7ms |
| hunterx20.jactr | 160 | 2917 | 20.6ms |

Tabuľka 5.2: Výkon jACT-R pri zrušení obmedzenia dĺžky cyklu

Treba podotknúť, že tento experiment testuje len procedurálny modul. Práca ostatných modulov by principiálne mala bežať v inom vlákne a teda by mali byť spracovateľné aj iným jadrom procesoru. Pri zrušení obmedzenia cyklu vlákno procedurálneho modulu plne vyťažilo jedno jadro. Experimentálne namerané hodnoty sú uvedené v tabuľke 5.2.1.

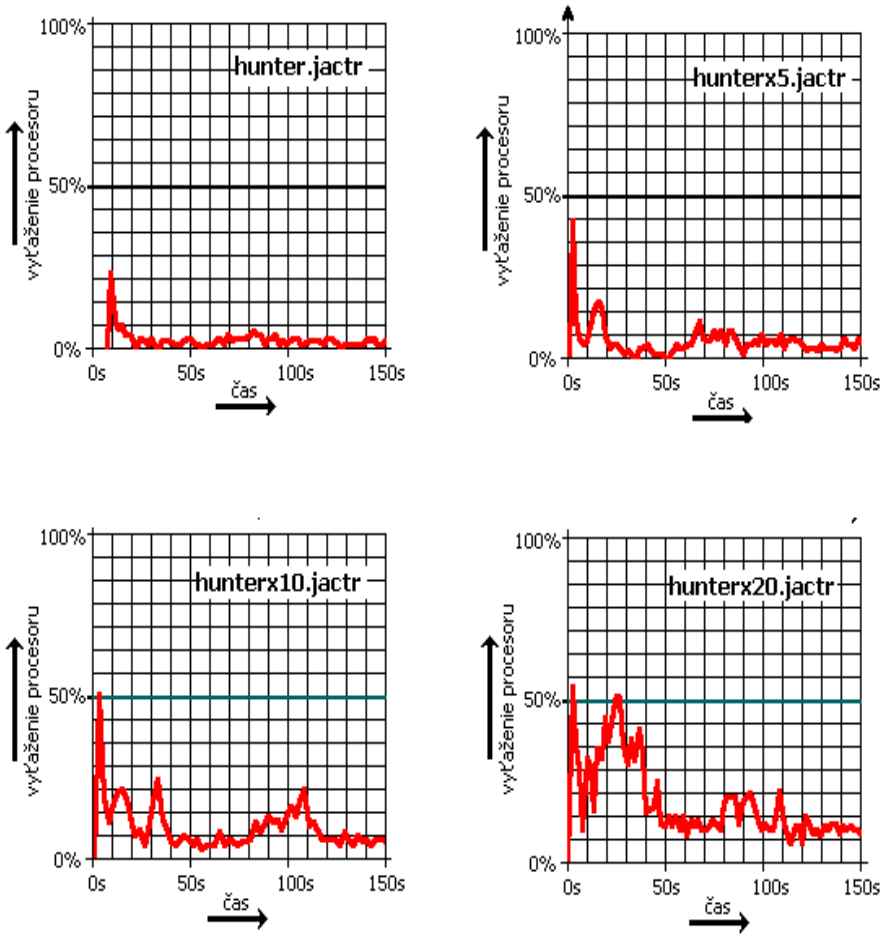
Namerané hodnoty sú zhodné s očakávanými hodnotami. Procedurálny modul pracuje v lineárnom čase podľa počtu pravidiel. To vidno pri 40,80 a 160 pravidlách, kedy počet cyklov aj dĺžka jedného cyklu lineárne rastie. Pri ôsmich pravidlách sa už príliš prejavuje výpočtový čas, ktorý sa spotrebováva medzi cyklami. Teoreticky zvládnuteľný počet pravidiel pri danom výkone počítača aby sa dodržal limit 50ms ne jeden cyklus je 400 pravidiel. Prakticky bude počet pravidiel nižší, pretože to môže ovplyvniť povaha pravidiel (počet podmienok a akcií) a výpočtová náročnosť jednotlivých modulov, s ktorými sa procedurálny modul delí o čas procesora.

### 5.2.2 Zaťaženie procesora

Tento experiment sa snaží ukázať zaťaženie procesora PoJACT-R agentom. Pre testovanie sa použili rovnaké modely ako v predchádzajúcom experimente. Merania boli prevedené pomocou utility AnVir Task Manager Pro[1]. Obrázok 5.1 ukazuje ako veľmi vyťažil proces agenta procesor pri rôznych počtoch pravidiel.

Je vidieť, že okrem času pri spustení agenta mal aj 160 pravidlový model ešte rezervu pre dodržiavanie 50ms cyklu. Je treba poukázať, že pri strete s nepriateľom, sa zaťaženie procesora zvyšuje, čo môže byť spôsobené väčším

[1]http://www.anvir.com/taskmanagerpro/




Obr. 5.1: Vyťaženie procesoru

množstvom spracovaných vnemov alebo komplikovanejším výberom samotnej akcie, keďže pri strete s nepriateľom je splnených viac prodmienok pravidiel. Pekne to vidno hlavne na grafe vyťaženia modelu hunterx10.jactr, ktorý sa stretol s nepriateľom medzi 30-40s a medzi 90-110s.

## 5.3 Zhodnotenie rozšírenia PoJACT-R

Pred implementáciou bolo ťažké povedať, ako užitočný a funkčný PoJACT-T bude. Vyzerá to tak, že funkčný v podstate je. Dajú sa cez neho riadiť agenti pomocou teórie ACT-R. Predchádzajúca podkapitola ukázala, že výkonnostne tiež na tom nie je zle. Navyše vďaka modularite a bufferom je možné využívať viac vlákien a teda aj viac jadier procesoru. Ďalej treba podotknúť, že PoJACT-R je limitovaný simulátorom jACT-R. To znamená, že dokáže len to z ACT-R teórie, čo je implementované v jACT-R. Pred ľubovolým výskumom na PoJACT-R je dobré skontrolovať, či je daný aspekt ACT-R už zahrnutý v jACT-R.

Pri porovnaní PoJACT-R agenta—lovca so štandardným Pogamut agentom—lovcom, ktorý používa jednoduché if-then pravidlá vidno, že správaním sú v podstate rovnaký. Navyše Pogamut agent—lovec využíva Pogamut navigáciu a inventár, a teda sa v priestore lepšie orientuj a lepšie narába so zbraňanmi. Na mieste je teda otázka načo je vlastne PoJACT-R a riadenie pomocou ACT-R dobré. Odpoveďou je primárna cieľová skupina, ktorej je PoJACT-R určený a to kognitívny výskumníci. Tých zaujíma plauzibilita a psychologické pozadie viac ako využiteľnost v praxi. To neznamená, že PoJACT-R je v hernom priemysle nepoužiteľný. Momentálne je PoJACT-R len prvým krokom a sú potrebné ďaľšie experimenty, aby sa dalo povedať viac.

# Kapitola 6

# Záver

Cieľom rozšírenia PoJACT-R bolo prepojiť implementáciu kognitívnej architektúry ACT-R s platformou Pogamut a odpovedať na otázky položené v úvode. Tento cieľ sa podarilo naplniť a na otázky do značnej miery odpovedať. PoJACT-R skutočne dovoľuje riadiť UT2004 agentov skrze Pogamut pomocou simulátoru jACT-R založenom na teórii ACT-R. Jadro prepojenia a časti závislé na UT2004 sú oddelené. Typová bezpečnosť sa využíva všade, kde to nie je kontraproduktívne. Boli navrhnuté riešenia pre zjednodušenie použitia, ako napríklad editor modelov. A boli ukázané limity plauzibility, ako aj jedno riešenie pre zníženie dopadu týchto limít. Navyše boli implementované percepčno-motorové ACT-R moduly pre UT2004 a ukážkový model agenta—lovca.

Cieľovou skupinou PoJACT-R sú kognitívny výskumníci. Zásadnou limitáciou pre túto skupinu je simulátor jACT-R, ktorý nemusí držať krok s teóriou ACT-R. Použitie v hernom priemysle je teoreticky možné, ale bude potrebný další výskum a vývoj kým bude prakticky realizovateľné. Motiváciou pre tento výskum je viacvláknová podstata jACT-R a teda možnosť využitia viacerých jadier procesoru na riadenie agenta.

Na PoJACT-R je stále na čom pracovať. Z implementačných úloh treba spomenúť vytvorenie komplexného rozšírenia IDE, s ktorým by zjednodušil použitie tak, aby sa dal ACT-R využívať pri výuke študentov. Inou možnosťou je navrhnutie, implementovanie a skúmanie plauzibilných percepčno-motorových modulov. Zaujímavou prácou by bolo navrhnutie ACT-

R modelu pre schovávačku[1] na dopredu neznámej mape.

Práca tiež môže poslúžiť pre záujemcov o Pogamut ako úvod k programovaniu a pre záujemcov o ACT-R ako úvod ku komplexnejším prácam o ACT-R.

[1] Anglicky Hide and seek

# Literatúra


[1] Duchi, J. C. , Laird, J. E., *Creating Human-like Synthetic Characters with Multiple Skill Levels: A Case Study using the Soar Quakebot* , AAAI Press 2000, (2000) 75-79.

[2] Best, B. J., Lebiere, C., *Cognitive agents interacting in real and virtual worlds.* , Cambridge University Press, New York, (2006) 186-218.

[3] Stenberg, R. J., *Kognitivní psychologie.* , Portál, Praha, (2002).

[4] Anderson, J. R., *How can the human mind occur in the physical universe?*, Oxford University Press, New York, (2007).

[5] Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., Qin, Y., *An integrated theory of the mind* , Psychological Review 111, (2004) 1036-1060.

[6] Fodor, J. A., *The Modularity of Mind* , MIT/Bradford Books, Cambridge, (1983).

[7] Franklin, S., *Artificial Minds* , MIT Press, Cambridge, (1995).